# *eunix: echo*

*Eric Bailey*

*October 31, 2017* [1]

[1] Last updated February 26, 2024

Reimplementations of `echo` in C and Rust for my own edification.

## *Contents*

## C

2a ⟨*src/echo.c* 2a⟩≡
⟨*Include headers.* 2c⟩

⟨*Forward declarations.* 2e⟩

⟨*Define the* `main` *function.* 2b⟩

⟨*Define the* `usage` *function.* 2f⟩

Root chunk (not used in this document).

### The `main` Function

2b ⟨*Define the* `main` *function.* 2b⟩≡

```
int main(int argc, char *argv[])
{
    ⟨Process given options. 3a⟩

    ⟨Print each string, separated by a space. 4d⟩

    ⟨Print a newline unless the -n option was given. 3d⟩

    return 0;
}
```

This code is used in chunk 2a.
Defines:
`argc`, used in chunk 4.
`argv`, used in chunk 4.

### Include Headers

Include the GNU `getopt` function from the GNU C Library.

"The `getopt` function gets the next option argument from the argument list specified by the `argv` and `argc` arguments. Normally these values come directly from the arguments received by `main`." – GNU, 2017

2c ⟨*Include headers.* 2c⟩≡

```
#include <getopt.h>
```

This definition is continued in chunk 2d.
This code is used in chunk 2a.
Defines:
`getopt`, used in chunk 4a.
`opterr`, used in chunk 3a.
`optind`, used in chunks 3f and 4d.
`optopt`, used in chunk 4a.

Include the core input and output functions from the C standard library.

2d ⟨*Include headers.* 2c⟩+≡

```
#include <stdio.h>
```

This code is used in chunk 2a.
Defines:
`EOF`, used in chunk 4a.
`printf`, used in chunks 2f and 4c.
`putchar`, used in chunks 3d and 4b.

### The `usage` Function

Define the `usage` function, which displays information about how to use `echo`, including ⟨*known options* 3b⟩.

2e ⟨*Forward declarations.* 2e⟩≡

```
void usage();
```

This code is used in chunk 2a.

2f ⟨*Define the* `usage` *function.* 2f⟩≡

```
void usage()
{
    printf("Usage: echo [-n] [string ...]\n");
}
```

This code is used in chunk 2a.
Uses `printf` 2d.

### *Processing Options*

Set `opterr` to `0` to tell `getopt` not to print an error message upon encountering un⟨*known options* 3b⟩.

3a ⟨*Process given options.* 3a⟩≡
```
opterr = 0;
```

This definition is continued in chunk 3.
This code is used in chunk 2b.
Uses `opterr` 2c.

`echo` accepts `-n` and prints other options.

3b ⟨*known options* 3b⟩≡
```
n
```

This code is used in chunk 4a.

Declare a variable `newline_flag` to determine whether or not to print a newline after printing the rest of the given strings.

3c ⟨*Process given options.* 3a⟩+≡
```
int newline_flag = 1;
```

This code is used in chunk 2b.

By default, print a trailing newline.

3d ⟨*Print a newline unless the* `-n` *option was given.* 3d⟩≡
```
if (newline_flag)
    putchar('\n');
```

This code is used in chunk 2b.
Uses `putchar` 2d.

When the `-n` option is given, set `newline_flag` to `0`, thereby disabling the printing of the trailing newline.

3e ⟨*Handle* `-n`. 3e⟩≡
```
case 'n':
    newline_flag = 0;
    break;
```

This code is used in chunk 3g.

If the user gives an unknown option, i.e. one not included in the ⟨*known options* 3b⟩, decrement `optind` by `1` in order to print it later.

"This variable is set by `getopt` to the index of the next element of the `argv` array to be processed." – GNU, 2017

3f ⟨*Handle unknown options.* 3f⟩≡
```
case '?':
    optind-;
    break;
```

This code is used in chunk 3g.
Uses `optind` 2c.

Loop through given options and handle them appropriately.

3g ⟨*Process given options.* 3a⟩+≡
```
int c;

while (⟨Process known options until EOF. 4a⟩) {
    switch (c) {
    ⟨Handle -n. 3e⟩
    ⟨Handle unknown options. 3f⟩
    }
}
```

This code is used in chunk 2b.

Stop processing options when optopt is set. Otherwise, process each known option as c until EOF.

4a ⟨*Process known options until* EOF. 4a⟩≡
```
optopt == '?' && (c = getopt(argc, argv, "⟨known options 3b⟩")) ≠ EOF
```
This code is used in chunk 3g.
Uses argc 2b, argv 2b, EOF 2d, getopt 2c, and optopt 2c.

"When getopt encounters an unknown option character... it stores that option character in this variable." – GNU, 2017

4b ⟨*print a space* 4b⟩≡
```
putchar(' ');
```
This code is used in chunk 4f.
Uses putchar 2d.

## *Echoing Strings*

Loop through argv, starting at optind, and ⟨*print a space* 4b⟩ between each string.

4c ⟨*Print the current string.* 4c⟩≡
```
printf("%s", argv[index]);
```
This code is used in chunk 4d.
Uses argv 2b, index 4d, and printf 2d.

4d ⟨*Print each string, separated by a space.* 4d⟩≡
```
for (int index = optind; index < argc; index++) {
    ⟨Print the current string. 4c⟩
    ⟨Print a space unless the current string is the last argument. 4f⟩
}
```
This code is used in chunk 2b.
Defines:
index, used in chunk 4.
Uses argc 2b and optind 2c.

4e ⟨*the current string is not the last argument* 4e⟩≡
```
index < argc - 1
```
This code is used in chunk 4f.
Uses argc 2b and index 4d.

If index is less than argc - 1 then ⟨*the current string is not the last argument* 4e⟩, so ⟨*print a space* 4b⟩.

4f ⟨*Print a space unless the current string is the last argument.* 4f⟩≡
```
if (⟨the current string is not the last argument 4e⟩)
    ⟨print a space 4b⟩
```
This code is used in chunk 4d.

*Full Listing*

src/echo.c

```c
#include <getopt.h>
#include <stdio.h>

void usage();

int main(int argc, char *argv[])
{
    opterr = 0;

    int newline_flag = 1;

    int c;

    while (optopt == '?' && (c = getopt(argc, argv, "n")) != EOF) {
        switch (c) {
        case 'n':
            newline_flag = 0;
            break;
        case '?':
            optind--;
            break;
        }
    }

    for (int index = optind; index < argc; index++) {
        printf("%s", argv[index]);
        if (index < argc - 1)
            putchar(' ');
    }

    if (newline_flag)
        putchar('\n');

    return 0;
}

void usage()
{
    printf("Usage: echo [-n] [string ...]\n");
}
```

## Rust

### The main function

6b ⟨*Define the* `main` *function.* 6b⟩≡

```
fn main() {
    ⟨Parse Args. 6e⟩

    ⟨Print strings separated by a space unless no_space is set. 7a⟩

    ⟨Print a newline unless no_newline is set. 7b⟩
}
```

This code is used in chunk 6a.

6a ⟨*src/bin/echo.rs* 6a⟩≡

```
⟨Use the clap crate. 6c⟩

⟨Define the Args struct. 6d⟩

⟨Define the main function. 6b⟩
```

Root chunk (not used in this document).

### Dependencies

Describe clap

6c ⟨*Use the clap crate.* 6c⟩≡

```
extern crate clap;

use clap::Parser;
```

This code is used in chunk 6a.

### Handling `Args`

6d ⟨*Define the Args struct.* 6d⟩≡

```
#[derive(Parser)]
#[command(trailing_var_arg = true, allow_hyphen_values = true)]
#[clap(disable_help_flag = true, disable_version_flag = true)]
struct Args {
    /// Do not output a newline
    #[arg(short = 'n', long)]
    no_newline : bool,

    /// Do not separate arguments with spaces
    #[arg(short = 's', long)]
    no_space : bool,

    string : Vec<String>,
}
```

This code is used in chunk 6a.
Defines:
`Args`, used in chunk 6e.
`no_newline`, used in chunk 7b.
`no_space`, used in chunk 7a.

6e ⟨*Parse* `Args`. 6e⟩≡

```
let args = Args::parse();
```

This code is used in chunk 6b.
Uses `Args` 6d.

### *Echoing strings*

7a ⟨*Print strings separated by a space unless no_space is set.* 7a⟩≡

```
let strings : Vec<_> = args.string;

strings.iter().enumerate().for_each(|(i, s)| {
    print!("{}", s);

    if !args.no_space && i < strings.len() - 1 {
        print!(" ");
    }
});
```

This code is used in chunk 6b.
Uses `no_space` 6d.

7b ⟨*Print a newline unless no_newline is set.* 7b⟩≡

```
if !args.no_newline {
    println!()
}
```

This code is used in chunk 6b.
Uses `no_newline` 6d.

*Full Listing*

src/bin/echo.rs

```rust
extern crate clap;

use clap::Parser;

#[derive(Parser)]
#[command(trailing_var_arg = true, allow_hyphen_values = true)]
#[clap(disable_help_flag = true, disable_version_flag = true)]
struct Args {
    /// Do not output a newline
    #[arg(short = 'n', long)]
    no_newline : bool,

    /// Do not separate arguments with spaces
    #[arg(short = 's', long)]
    no_space : bool,

    string : Vec<String>,
}

fn main() {
    let args = Args::parse();

    let strings : Vec<_> = args.string;

    strings.iter().enumerate().for_each(|(i, s)| {
        print!("{}", s);

        if !args.no_space && i < strings.len() - 1 {
            print!(" ");
        }
    });

    if !args.no_newline {
        println!()
    }
}
```

## *Chunks*



## *Index*



## *References*

GNU. The GNU C Library: Using the getopt function. https://www.gnu.org/software/libc/manual/html_node/Using-Getopt.html, 2017. Accessed: 2017-11-05.

*To-Do*